**Warm-Up Polynomial Class Part B**

For Part B of the warm-up, you are to create a polynomial class in Java using a linked structure

The polynomial class should consist of a linked list of nodes accessed through *head*.  
You will need to create a class of *terms* (Objects representing the individual terms of the polynomial)



The polynomial class will consist of linked nodes:

class polynomial{  
private node head;  
private int degree;

The *node* (a *node* class in the same folder/package):  
*protected terms* term;  
*protected node* next;

The *terms* (in the same folder/package) will consist of*:*  
*protected double coeff;  
protected int power;*

And *member methods:*constructor(s)  
evaluate(x)   
add- which will add a polynomial q to the given polynomial p = p+q  
subtract – which will subtract a polynomial q from the given polynomial p = p-q  
scale = which will multiply the given polynomial by a constant a p =ap  
multiply – which will multiply the given polynomial by a polynomial q p = p\*q  
  
and *polynomial methods:*sum – which adds 2 polynomials and creates a new polynomial   
 without destroying the original r = p+q  
diff - which subtracts 2 polynomials and creates a new polynomial   
 without destroying the originals r = p-q  
product - which multiplies 2 polynomials and creates a new polynomial   
 without destroying the original r = p\*q



The polynomial should maintain the terms in order, sorted by the power. (*hint: use the insertSorted* method we designed in lecture, with the “order” using term.power.)

You do not need to use the LinkedList class to do this project! You are just using the linked-structure style to manage the polynomials.

Again, you do not need to submit a main for this part of the Warm-Up. The idea is to demonstrate the ability to use the linked structure to manage data, and to be able to discuss (Part C) the relative strengths and weaknesses of using arrays and links.

package shapesParser;

public class Polynomials {

private Node head;

private int degree;

public Polynomials() {

}

public int degree(){

if(head == null) {

System.out.println("List is empty");

return 0 ;

}

return (int)head.term.coeff;

}

public float evaluate(float x) {

float answer = 0;

for (Node p1 = this.head; p1 != null; p1 = p1.next){

answer += p1.term.coeff \* (Math.pow(x, p1.term.power));

}

return answer;

}

//add- which will add a polynomial q to the given polynomial p = p+q

public void add(Polynomials p2)

{

Polynomials sum=new Polynomials();

Node a=this.head;

Node b=p2.head;

while(a!=null || b!= null) {

if((a!=null && b!=null)) {

if(a.term.power < b.term.power) {

sum.insertSorted(new Terms(a.term.coeff, a.term.power));

a=a.next;

}

else if(a.term.power > b.term.power) {

sum.insertSorted(new Terms(b.term.coeff,b.term.power));

b=b.next;

}

else {

sum.insertSorted(new Terms(a.term.coeff+b.term.coeff, a.term.power));

a=a.next;

b=b.next;

}

}

else {

if(a==null) {

sum.insertSorted(new Terms(b.term.coeff,b.term.power));

b=b.next;

}

else {

sum.insertSorted(new Terms(a.term.coeff,a.term.power));

a=a.next;

}

}

}

this.degree = sum.degree;

this.head = sum.head;

}

public void subtract( Polynomials p2)

{

Polynomials sum=new Polynomials();

Node a=this.head;

Node b=p2.head;

while(a!=null || b!= null) {

if((a!=null && b!=null)) {

if(a.term.power < b.term.power) {

sum.insertSorted(new Terms(a.term.coeff, a.term.power));

a=a.next;

}

else if(a.term.power > b.term.power) {

sum.insertSorted(new Terms(b.term.coeff,b.term.power));

b=b.next;

}

else {

sum.insertSorted(new Terms(a.term.coeff-b.term.coeff, a.term.power));

a=a.next;

b=b.next;

}

}

else {

if(a==null) {

sum.insertSorted(new Terms(b.term.coeff,b.term.power));

b=b.next;

}

else {

sum.insertSorted(new Terms(a.term.coeff,a.term.power));

a=a.next;

}

}

}

this.degree = sum.degree;

this.head = sum.head;

}

//scale = which will multiply the given polynomial by a constant a p =ap

public void scale(int x) {

Polynomials pol=new Polynomials();

for(Node a=this.head;a!=null;a=a.next) {

pol.insertSorted(new Terms(a.term.coeff\*x, a.term.power));

}

this.degree = pol.degree;

this.head = pol.head;

return;

}

//multiply – which will multiply the given polynomial by a polynomial q p = p\*q

public void multiply(Polynomials p2) {

Polynomials pol=new Polynomials();

for(Node b=p2.head;b!=null;b=b.next) {

Polynomials temp=new Polynomials();

for(Node a=this.head;a!=null;a=a.next) {

temp.insertSorted(new Terms(a.term.coeff\*b.term.coeff, a.term.power+b.term.power));

}

pol=sum(pol,temp);

}

this.degree = pol.degree;

this.head = pol.head;

}

//<------------------------------------------------------------------------------------------------>

//sum – which adds 2 polynomials and creates a new polynomial

public Polynomials sum(Polynomials p1,Polynomials p2)

{

Polynomials sum=new Polynomials();

Node a=p1.head;

Node b=p2.head;

while(a!=null || b!= null) {

if((a!=null && b!=null)) {

if(a.term.power < b.term.power) {

sum.insertSorted(new Terms(a.term.coeff, a.term.power));

a=a.next;

}

else if(a.term.power > b.term.power) {

sum.insertSorted(new Terms(b.term.coeff,b.term.power));

b=b.next;

}

else {

sum.insertSorted(new Terms(a.term.coeff+b.term.coeff, a.term.power));

a=a.next;

b=b.next;

}

}

else {

if(a==null) {

sum.insertSorted(new Terms(b.term.coeff,b.term.power));

b=b.next;

}

else {

sum.insertSorted(new Terms(a.term.coeff,a.term.power));

a=a.next;

}

}

}

return sum;

}

public Polynomials minus(Polynomials p1,Polynomials p2)

{

Polynomials sum=new Polynomials();

Node a=p1.head;

Node b=p2.head;

while(a!=null || b!= null) {

if((a!=null && b!=null)) {

if(a.term.power < b.term.power) {

sum.insertSorted(new Terms(a.term.coeff, a.term.power));

a=a.next;

}

else if(a.term.power > b.term.power) {

sum.insertSorted(new Terms(b.term.coeff,b.term.power));

b=b.next;

}

else {

sum.insertSorted(new Terms(a.term.coeff-b.term.coeff, a.term.power));

a=a.next;

b=b.next;

}

}

else {

if(a==null) {

sum.insertSorted(new Terms(b.term.coeff,b.term.power));

b=b.next;

}

else {

sum.insertSorted(new Terms(a.term.coeff,a.term.power));

a=a.next;

}

}

}

return sum;

}

public Polynomials product(Polynomials p1,Polynomials p2) {

Polynomials pol=new Polynomials();

for(Node b=p2.head;b!=null;b=b.next) {

Polynomials temp=new Polynomials();

for(Node a=p1.head;a!=null;a=a.next) {

temp.insertSorted(new Terms(a.term.coeff\*b.term.coeff, a.term.power+b.term.power));

}

pol=sum(pol,temp);

}

return pol;

}

public void insertSorted(Terms term) {

Node new\_node = new Node(term);

Node current;

if (head == null || head.term.power>= new\_node.term.power) {

new\_node.next = head;

head = new\_node;

}

else {

current = head;

while (current.next != null && current.next.term.power < new\_node.term.power)

current = current.next;

new\_node.next = current.next;

current.next = new\_node;

}

this.degree = degree();

}

public void display() {

Node temp;

for (temp = head; temp != null; temp = temp.next) {

if (temp.next != null) {

System.out.print(temp.term.coeff + "x ^ " + temp.term.power + " + ");

} else {

System.out.println(temp.term.coeff);

}

}

}

public static void main( String[] args ){

Polynomials pol = new Polynomials();

pol.insertSorted(new Terms(1,3));

pol.insertSorted(new Terms(1,3));

pol.insertSorted(new Terms(1,3));

pol.display();

Polynomials pol2 = new Polynomials();

pol2.insertSorted(new Terms(10,3));

pol2.insertSorted(new Terms(10,3));

pol2.insertSorted(new Terms(12,3));

// Add

pol.add(pol2);

pol.display();

//substract

//pol2.subtract(pol);

//pol2.display();

//multiply

//pol2.multiply(pol);

//pol2.display();

//scale

//pol2.scale(5);

//pol2.display();

//System.out.println("hello");

}

}